

## **METHODS AND APPARATUS FOR CREATION OF PARSING RULES**

### **Cross Reference to Related Application**

The present application is related to the U.S. patent application identified as Serial No. 10/334,254 (attorney docket no. RSW8-2002-0328), filed on October 23,  
5 2002, and entitled "Smart Event Parser Using Self-Learning and Self-Configuration," the disclosure of which is incorporated by reference herein.

### **Field of the Invention**

The present invention relates generally to the field of autonomic computing and, more particularly, to the generation of parsing rules for use in a rule-based system such as  
10 an event message adaptation system.

### **Background of the Invention**

Effective management of event messages is the cornerstone of high quality information technology (IT) service delivery.

Intense competition among IT service providers to demonstrate high quality  
15 service management (e.g., low response times, high availability) has led to very aggressive goals for IT-based services. Realizing these goals requires proactive management processes which provide early detection and isolation of IT event messages signaling service delivery problems. As IT service providers are forced by an extremely competitive market to aggressively control cost of service delivery, the automation of  
20 these processes becomes increasingly critical. This capability of automated event detection, problem isolation and resolution is a key aspect of an autonomic computing strategy. This is especially the case for complex IT systems comprising distributed, heterogeneous components.

As is known, "autonomic computing" is a comprehensive and holistic approach to  
25 self-managed computing systems with a minimum of human interference, e.g., see P.

Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," IBM Research, October 2001, the disclosure of which is incorporated by reference herein.

5 Real-time, high-performance event management systems universally require transformation of the incoming event data to a common format prior to application of event processing logic. This transformation from unique formats to a common format is controlled by parsing rules.

10 Creation of parsing rules that transform event data into a unified format has traditionally been a very time consuming exercise that requires technology domain experts to develop unique parsing rules for all event messages. In the past, parsing has often been addressed manually by creating ad-hoc parsers directed to event logs of specific technologies and applications.

15 Several problems exist with such an approach. First, the manual approach involves a time-consuming, error prone process. Second, the manual approach requires a user to have both: (1) domain knowledge in understanding data formats; and (2) programming knowledge in translating domain knowledge into event data parsing rules.

In addition, the manual approach has been rendered ineffective by significant challenges emerging from the present day IT environment.

20 A critical challenge in the deployment of autonomic event management methods and systems is the need for the solution to address very large numbers of events in real-time, support a broadening spectrum of event message formats, and recognize and process individually thousands of unique event messages.

25 The most onerous issue is event volume. Many IT operations centers report volumes of one million or more events per day. More IT users are reaching that plateau each month. Unfortunately, users lack a process for collection, parsing and extraction of pertinent event data which effectively addresses this scaling issue.

The IT industry has introduced a broad range of proprietary and standardized event protocols, log file formats, and (even within a single protocol) syntax. The variety of formats assumed by event messages adds considerable complexity to the event data environment of the user. Viewed from a practical data management perspective, the  
5 variety in event formats will add significantly to the effort the customer will be required to invest in development of data parsing rules.

Further, the torrent of events generated across the IT environment of the user is composed of thousands of unique event types, each containing potentially important management information and each, potentially, requiring unique parsing rules.

10 To summarize, many users contend with more than a million event messages per day. Their event streams contain a multitude of differing data protocols and formats. The individual events within these event streams represent thousands of unique event types. Traditional labor intensive approaches to the parsing analysis of this mass of event data are inadequate.

15 Thus, a need exists for parsing rule creation techniques that are supported with automated facilities such that the above-mentioned and other limitations may be overcome.

### **Summary of the Invention**

20 The present invention provides techniques for parsing rule creation that are supported with automated facilities such that the above-mentioned and other limitations may be overcome. Advantageously, the invention allows a system implementing such techniques to realize gigabyte data reduction.

25 In a first illustrative aspect of the invention, a technique for constructing one or more message parsing rules comprises the following steps. First, message data representing past messages, for example, associated with a network, an application and/or a system being analyzed, is obtained. For example, this may involve reading the past or

historical message data from messages logs or having a system point to the message data in existing data storage. Parsing rules are then generated by a process from one or more existing rule templates and/or based on user selection and classification of at least a portion of a message. For example, the user may choose a message part and  
5 demonstratively classify the part, for example, as a positive or negative example. The generated rules may then be stored for access by a rule-based parsing system such as a message adaptation system.

Prior to generation of the one or more parsing rules, a message structure may be established upon which generation of the rules may be based. Thus, in a second  
10 illustrative aspect of the invention, when one or more previously generated templates are available, the step of establishing a message structure may comprise the following steps. First, a skeleton of the message may be created. A skeleton may, for example, contain information about message start, message end, separation between fields, and some additional information about the message. Next, previously generated templates may be  
15 matched against the message skeleton. Then, possible matches may be provided to the analyst for validation and choice of proper message structure. Next, if the structure of the message is found to be insufficient, templates may be built by an iterative process between analyst (human) and machine (computer system) based on the analyst's choice of a part of the message and possibly additional demonstrative classification of the  
20 chosen part as a positive or negative example. Lastly, the approved message structure may be output as a possible message structure template.

In a third illustrative aspect of the invention, the step of building parsing rules iteratively by demonstration, possibly based on positive or negative examples, may comprise the following steps. First, a machine may parse message data sequentially until  
25 it encounters the end of the data or an unparseable message. An unparseable message may be displayed in a log viewer. Then, the analyst may define an example, e.g., the analyst selects part of the message, possibly comprising multiple segments, and marks

the selected part as a positive or negative example. Next, the machine may learn based on the example, e.g., the machine may create possible rules based on rule templates, a knowledge base, and the output message structure, covering positive examples but not containing negative examples and shows the created rule templates to the analyst in the form of a priority list. The analyst may choose from templates and define a mapping based on the output structure. Next, the machine may refine and verify the rule. The rule may then be added to the parsing rules and run against all data. Parsing results or parsing errors may be shown to the analyst. Lastly, the analyst may make a final decision, e.g., the analyst accepts or rejects the rule. An accepted rule may be added to the parsing rules. These steps may be repeated until all messages are parsed without errors.

These and other objects, features and advantages of the present invention will become apparent from the following detailed description of illustrative embodiments thereof, which is to be read in connection with the accompanying drawings.

#### **Brief Description of the Drawings**

FIG. 1 is a block diagram illustrating a rule building system and a message adaptation system with which the rule building system may be implemented according to an embodiment of the present invention;

FIG. 2 is a diagram illustrating an exemplary rule building application mark-up according to an embodiment of the present invention;

FIG. 3 is a control flow diagram of rule generation according to an embodiment of the present invention;

FIG. 4 is a diagram illustrating a message structure schema according to an embodiment of the present invention;

FIG. 5 is a flow diagram illustrating a methodology of establishing message structure according to an embodiment of the present invention;

FIG. 6 is a flow diagram illustrating a methodology of building parsing rules by demonstration using positive and negative examples according to an embodiment of the present invention;

FIG. 7 is a diagram illustrating a message output schema according to an  
5 embodiment of the present invention; and

FIG. 8 is a block diagram illustrating a generalized hardware architecture of a computer system suitable for implementing a rule building system according to the present invention.

#### **Detailed Description of Preferred Embodiments**

10 The present invention will be described below in the context of an exemplary message adaptation system. However, it is understood that the invention is not limited to use with a message adaptation system but is rather more generally applicable for use in accordance with any rule-based parsing system in which it is desirable to provide automated parsing rule construction capabilities.

15 Referring initially to FIG. 1, a block diagram illustrates a parsing rule building system and a message adaptation system with which the rule building system may be implemented according to an embodiment of the present invention. As shown, dotted line A serves to demarcate the rule building system from a computing system using a message adaptation system, i.e., the rule building system is below line A and the  
20 computing system using the message adaptation system is above line A.

Thus, as depicted in FIG. 1, items 150, 151, 152, 160, 161 and 162 represent a distributed computing system in which message adapter 140 operates by translating logs of software applications 150, 151 and system 152 into a common format understood by message consumers 160, 161 and 162 according to the subscription procedure of the  
25 message consumers. In order to operate, message adapter 140 uses parsing rules stored in rule storage 170.

In general, rule builder 100 builds parsing rules offline for the distributed computing system. It is to be understood that the techniques employed by rule builder 100 may interact with the distributed computing system in two ways.

First, rule builder 100 reads historical logs 110 provided by applications 150, 151  
5 and/or system 152, or accesses message data directly from applications 150, 151 and/or system 152. Rule builder 100 uses rule template storage 120 to store rule templates for present and future rule construction. Thus, rule builder 100 extracts rule templates from rule template storage 120 at the beginning of the parsing rule construction process and stores newly created rules at the end of the rule construction process. Results of the  
10 parsing rule construction process are stored in parsing rules file 130 and then transferred to rule storage 170. The stored rules are then used online by message adapter 140 in the applications and system logs translation process such that event data is translated into a common format understood by the message consumers.

Referring now to FIG. 2, a diagram illustrates an example of a rule building  
15 application mark-up or graphical user interface (GUI) for use in accordance with rule builder 100 (FIG. 1). More specifically, the figure illustrates panels (viewers) used by rule builder 100 to construct parsing rules in accordance with user interaction and feedback.

Panel 210 displays the current message structure (described below in the context  
20 of FIG. 4) which contains message start, message end, separator and other information useable to identify individual messages.

Panel 220 lists currently active parsing rules according to the output structure (described below in the context of FIG. 7). New rules according to the output structure can be added into the list and existing rules can be edited or removed from the list by an  
25 analyst (user).

Panel (log viewer) 230 displays one message, as defined by the message structure, and allows an analyst to select a part of the message composed of, for example, multiple

segments in order to describe an example to be classified as positive or negative to construct matching pattern templates 260. Selection of a message segment may be performed, by way of example only, by the user changing font types, font sizes, font colors, font styles and/or background colors, and/or by adding cross-out lines or underlines, with respect to the text in the message segment. The matching pattern templates can be (but are not limited to) regular expressions or position-based descriptions of the message segments.

Panel (rule building view) 240 presents the parsing rule currently under construction, which may include a machine pattern and a transformation rule. A transformation rule specifies the method of transforming the matched message segments to a normalized format by (but not limited to) selection, permutation, and/or assigning of a string constant (or input token) as output. The user can refine the rule manually.

Panel (result viewer) 250 shows the effect of the matching rule to the current message and the transformation rule and keeps them updated when the user changes the parsing rule. If one (or some) of the rules generates a parsing error, the parsing error information is displayed in result viewer 250.

Referring now to FIG. 3, a control flow diagram illustrates a systematic parsing rule construction methodology according to embodiment of the present invention. It is to be appreciated that the methodology depicted in FIG. 3 may be carried out by an analyst and a computer system (machine), or just by a computer system. Thus, as is evident, the methodology may be performed entirely in accordance with the machine (automated approach). However, the present invention realizes that benefits may be derived by providing use of the parsing rule construction system of the invention to a human expert (analyst or administrator) to systematically extract parsing rules from historical data (semi-automated approach).

It is to be noted that during the descriptions to follow, parenthetical reference will be made back to elements described above in the context of FIGs. 1 and 2.

In step 310, rule builder (100 of FIG. 1) loads historical data (110 of FIG. 1) into the log viewer (230 of FIG. 2). Historical data is data provided by applications (150, 151 of FIG. 1), a system (152 of FIG. 1) and/or a network. The historical data may be message data representing past messages associated with the network, the applications  
5 and/or the system being analyzed. For example, this may involve reading the past or historical message data from message logs or having a system (or application or network) point to the message data in existing data storage.

In step 320, a message structure is established. More particularly, an appropriate rule template for the message structure is established. Details of a process for  
10 establishing message structure are described below in the context of FIG. 5.

In step 330, the rule builder and analyst, based on his or her experience, build parsing rules by demonstration and classification of examples as positive or negative, i.e., by the user demonstrating to the machine what information (e.g., by way of classifying examples) the machine should use to generate parsing rules. More particularly, based on  
15 the message structure, parsing rules are generated by an iterative refinement process from existing templates or based on user choice of the message part and classification of the part as a positive or negative example. Details of a process for building the parsing rules is described below in the context of FIG. 6.

In step 340, built parsing rules are saved to a file (130 of FIG. 1) and passed to  
20 rule storage (170 of FIG. 1) to be used by a message adapter (140 of FIG. 1).

In step 350, built parsing rules are saved by the rule builder (100 of FIG. 1) as templates (120 of FIG. 1) for future parsing rule constructions.

FIG. 4 is a diagram illustrating a mapping in the XML (Extensible Markup Language) schema of a message structure description according to an embodiment of the  
25 present invention.

More particularly, FIG. 4 illustrates a description 400 of the message structure that may be used and/or stored as part of the parsing rules (130 of FIG. 1). The

description comprises attributes used in the message structure. Attributes may include, but are not limited to:

- (i) attribute "messageStart" which describes the start of the message in a unique way with the respect to given historical message data;
- 5       (ii) attribute "multiline" which takes value true when a message is represented in the historical data by multiple lines, and false if the message is represented in the historical data by a single line;
- (iii) attribute "eventEnd" is an optional attribute and it describes the end of a valid message in a unique way for the specific historical data;
- 10       (iv) attribute "separator" describes how different fields of the message are separated one from another for the specific historical log;
- (v) attribute "glueing" is optional and is used for multiline logs to combine a number of separate lines contributing to the message into one line for further rule construction;
- 15       (vi) attribute "msgType" helps to classify different logs on a high level, and may have values, for example, such as "TEC\_RECEPTION\_MSG", "DB2\_MSG", "DB2\_DIAG\_MSG", "WAS\_MSG", "WAS\_ACTIVITY\_MSG", and default value "UNKNOWN\_MSG".

Referring now to FIG. 5, a flow diagram illustrates a methodology of building or  
20 establishing message structure according to an embodiment of the present invention. FIG. 5 may be considered a detailed explanation of step 320 of FIG. 3. Thus, when one or more previously generated message structure templates are available, establishing message structure may comprises the following steps.

In step 501, the machine scans the historical message data, builds a skeleton of  
25 the message, and proposes a possible working message structure. A message skeleton may, for example, contain information about message start, message end, separation between fields, and some additional information about a message.

An example of a message skeleton and a template in the context of learning a separator in a message structure is as follow:

(1) the frequency of each character that can be considered as a separator is counted. Here, all special characters, such as “:”, “;” and a white (blank) space, are possible candidates for a separator.

(2) the candidate with the highest count is regarded as a separator. In our example, a space occurs the most. So, the methodology selects space as a separator. Similar mechanism can be developed for other parameters of the message structure.

Thus for the message:

“Jul 23 2003 05:49:30 somehostname TRIALINFO this is sample single line message”, the skeleton will be WSWWSWSWSWSWSWSWSWSWSW, where W stands for a word, S stands for the separator, and the value of the separator is one or more white spaces. Thus, the template in this case will be:

multiline = false

messageStart=^

Separator=\s+

Next, in step 502, the machine compares the proposed message structure with existing message structure templates. That is, previously generated message structure templates are matched against the message skeleton.

In step 503, the machine then selects a set of the most likely message structures and presents them to the analyst. That is, possible matches are provided to the analyst for validation and choice of proper message structure.

Next, in step 504, the analyst selects a message structure from the provided set of message structure templates, and the machine decides whether the current message structure contains enough information to identify individual messages.

If not, the machine prompts the analyst to provide positive or negative examples, in step 505, until enough information is gathered and a valid message structure is produced for use (step 506) in parsing rules construction (step 330 of FIG. 3).

5 That is, if the structure of the message is found to be insufficient, one or more message structure templates are built by an iterative process between analyst and machine based on analyst choice of part of the message, wherein the message possibly comprises multiple segments, and maybe based on additional classification of the chosen part as a positive or negative example. The approved message structure is output as a possible (candidate) message structure template.

10 Referring now to FIG. 6, a flow diagram illustrates a methodology of building parsing rules by demonstration using positive and negative examples according to an embodiment of the present invention. FIG. 6 may be considered a detailed explanation of step 330 of FIG. 3. The following steps may be repeated until whole historical message data is parsed without parsing errors, and the analyst finds the result of the parsing process satisfactory.

15 In step 601, the machine parses messages until it encounters a message generating error during parsing (i.e., the machine is unable to parse a message), or until it reaches the end of the data. The unparseable message is displayed to the analyst in the log viewer (230 of FIG. 2).

20 Next, in step 602, the analyst defines examples to be used for purposes of learning. The analyst selects part of the message (possibly comprising multiple segments) to be considered as an example. Next, the analyst classifies selection as a positive or a negative example.

25 In step 603, the machine learns the example. Traditional machine learning techniques may be employed such as, for example, those disclosed in "Discovery of Frequent Episodes in Event Sequences, Data Mining and Knowledge Discovery, 1(3), 1997; "Mining Association Rules Between Sets of Items in Large Databases," VLDB, pp.

207-216, 1993; "Mining Sequential Patterns: Generalization and Performance Improvements," Proc. of the Fifth Int'l Conference on Extending Database Technology, Avignon, France, 1996; and "Machine Learning," Tom Mitchell, 1997, the disclosures of which are incorporated by reference herein.

5           Thus, in accordance with one or more machine learning techniques, the machine may create possible (candidate) parsing rule templates covering positive examples, but not including negative examples. Possible parsing rule templates are shown in panel 260 of FIG. 2. More particularly, the machine may create possible rules based on rule templates, a knowledge base, and the output message structure, covering positive  
10       examples but not containing negative examples and show the created rule templates to the analyst in the form of a priority list.

          Next, in step 604, the analyst critiques and modifies, if necessary, the parsing rule templates. The analyst chooses templates most appropriate from the set of parsing rule templates provided by the machine. The chosen template is then shown in the rule  
15       building view (240 of FIG. 2). The analyst modifies, if necessary, the rule template to create the parsing rule. That is, the analyst may choose from templates and define a mapping based on the output structure.

          In step 605, the machine refines and verifies the parsing rule by applying the parsing rule to the current message. The result of the application is shown in the result  
20       view (250 of FIG. 2) in the form of either a result of application of the parsing rule or as a parsing error. In the case of no parsing error and if the analyst is satisfied with the result, the machine returns to step 601 to proceed with parsing. Further, in the case when the analyst is satisfied with results, the machine uses ( step 606) and saves parsing rules (see steps 340 and 350 of FIG. 3).

25           That is, in accordance with step 605, a newly created rule is added to the parsing rules and run against all data. Parsing results or parsing errors are shown to the analyst. The analyst makes the final decision, i.e., the analyst accepts or rejects the rule. An

accepted rule is added to the parsing rules. These steps are repeated until all messages are parsed without errors.

We now give an illustration of the process of learning by demonstration, in the context of the steps of FIG. 6, through a simple example in which a user only selects one message portion as a positive example and wishes to define a message type (“msgType” as mentioned above).

Step 601: a new message is shown in log viewer:

1054304804 3 Fri May 30 10:26:44 2003 sampleHost A Cisco Link Down trap  
received from enterprise cisco-stack;

Step 602: analyst selects (by bolding text, as shown) a part of message in log viewer:

1054304804 3 Fri May 30 10:26:44 2003 sampleHost A Cisco **Link Down** trap  
**received** from enterprise cisco-stack;

Step 603: machine provides multiple rules expressed as regular expressions:

(1) matching: (\w\*) Down (\w\*) received.\*; messageType: \$1\_down,  
(2) matching: (\w\*) (\w\*) (\w\*) received.\*; messageType: \$1\_\$2,  
(3) . . . . etc.;

Step 604: analyst critiques by choosing proper template (e.g., template associated with rule (1) above) and modifying it, if needed; and

Step 605: machine refines and verifies the rule by applying rule to the record (current message) and showing result (in this case, Link\_Down) in the result viewer.

FIG. 7 is a diagram illustrating a mapping in the XML schema of an output message structure description 700 according to an embodiment of the present invention.

The output message structure describes an element containing a set of the sub-elements with the name “Fields”. In addition, element “OutputStructure” contains the following attributes: attribute “separator” which is used for the description of the

fields' separator and attribute "hashing" which is used for description of hashing representation of an attribute – value format of the message. Each sub-element "Fields" corresponds to required or optional fields of the OutputStructure.

Element "Fields" is illustrated as having sub-elements of two types: "Groups" and "RuleAttribute". "Groups" sub-elements correspond to the sub-element that may have sub-elements of "Groups" type or "RuleAttribute" type and used for grouping together multiple sub-elements. "RuleAttribute" sub-elements correspond to the attributes that are expected to be in the output message. "Groups" and "RuleAttribute" are elements that are shown in panel 220 of FIG 2. "RuleAttCreationTime" is an illustration of the "RuleAttribute" and corresponds to the message timestamp attribute.

Referring now to FIG. 8, a block diagram is shown illustrating a generalized hardware architecture of a computer system suitable for implementing the various functional components/modules and methodologies of a parsing rule building system and a message adaptation system as depicted in the figures and explained in detail herein. That is, the computer system shown in FIG. 8 may be considered to be the "machine" with which an analyst interacts, as described above in detail. It is to be understood that the individual components/modules and methodologies of the systems may be implemented on one such computer system, or on more than one separate such computer system. Also, individual components of the system may be implemented on separate such computer systems. It is to be appreciated that the user may interact directly with the one or more computer systems implementing the systems. Alternatively, the user may employ a computer system in communication (e.g., via a remote or local network) with the one or more computer systems implementing the systems in order to interact with the systems.

As shown, the computer system may be implemented in accordance with a processor 810, a memory 820 and I/O devices 830, coupled via a suitable computer bus or network 840. It is to be appreciated that the term "processor" as used herein is

intended to include any processing device, such as, for example, one that includes a CPU (central processing unit) and/or other processing circuitry. The term “memory” as used herein is intended to include memory associated with a processor or CPU, such as, for example, RAM, ROM, a fixed memory device (e.g., hard drive), a removable memory device (e.g., diskette), flash memory, etc. In addition, the term “input/output devices” or “I/O devices” as used herein is intended to include, for example, one or more input devices (e.g., keyboard, mouse, etc.) for entering data (e.g., user selections and examples, etc.) to the processing unit, and/or one or more output devices (e.g., CRT display, printer, etc.) for presenting results (e.g., parsing rule generation results, parsing results, etc.) associated with the processing unit. For example, system user interfaces (e.g., FIG. 2) employed by the user may be realized through such I/O devices. It is also to be understood that the term “processor” may refer to more than one processing device and that various elements associated with a processing device may be shared by other processing devices.

Accordingly, software components including instructions or code for performing the methodologies of the invention, as described herein, may be stored in one or more of the associated memory devices (e.g., ROM, fixed or removable memory) and, when ready to be utilized, loaded in part or in whole (e.g., into RAM) and executed by a CPU.

Although illustrative embodiments of the present invention have been described herein with reference to the accompanying drawings, it is to be understood that the invention is not limited to those precise embodiments, and that various other changes and modifications may be made by one skilled in the art without departing from the scope or spirit of the invention.